Call Symput

Mastering the Art of Call SYMPUT in SAS: A Comprehensive Guide

The SAS `CALL SYMPUT` procedure is a powerful tool for dynamic macro variable creation and manipulation. Unlike the `%LET` statement, which creates macro variables within the macro language processor, `CALL SYMPUT` creates macro variables within the data step, allowing you to leverage data step processing to define your macro variables. This flexibility proves invaluable for creating dynamic and adaptive SAS programs, particularly in scenarios where variable names or values need to be determined during data processing. This article will provide a comprehensive overview of `CALL SYMPUT`, exploring its functionality, syntax, usage scenarios, and potential pitfalls.

Understanding the Syntax and Functionality

The core syntax of `CALL SYMPUT` is straightforward:

```sas
CALL SYMPUT('macro-variable-name', value);
```

Here:

'macro-variable-name': This is a character string representing the name of the macro variable you wish to create or modify. It must be enclosed in single quotes.
value: This is the value assigned to the macro variable. It can be a numeric value, a character string (also enclosed in single quotes), or a SAS variable's value.

Crucially, `CALL SYMPUT` operates within the data step context. This means the macro variable's value is determined by the current row being processed. This opens up possibilities for creating macro variables based on aggregated data, conditional logic, or even iterating through datasets.

Practical Examples: Unveiling the Power of CALL SYMPUT

Let's explore several practical applications to illuminate the utility of `CALL SYMPUT`.

Example 1: Creating a macro variable based on a dataset's summary statistic:

Suppose we have a dataset `sales` with a variable `sales_amount`. We want to create a macro variable `total_sales` containing the sum of `sales_amount`.

```sas
data _null_;
set sales end=eof;
sales_sum + sales_amount;
if eof then do;
call symput('total_sales', sales_sum);
end;
run;

%put The total sales are: &total_sales;
```

This code calculates the sum and then uses `CALL SYMPUT` to assign it to the macro variable `total_sales`. The `%PUT` statement then displays the value of this newly created macro variable.

Example 2: Dynamically creating macro variables based on conditional logic:

Imagine we want to create a macro variable `highest_region` indicating the region with the maximum sales.

```sas
proc sql noprint;
```

```
select region
into :highest_region
from sales
group by region
having sum(sales_amount) = max(sum(sales_amount));
quit;

%put The highest selling region is: &highest_region;
```

While this example uses `PROC SQL`, the principle remains the same. The result of the SQL query is assigned to the macro variable `highest_region` using the implicit `CALL SYMPUT` functionality of `PROC SQL`'s `INTO` clause.

Example 3: Iterating through a dataset to create multiple macro variables:

Let's say we have a dataset listing product names and their prices, and we want to create a macro variable for each product's price.

```sas
data _null_;
set products;
call symput(cats('price_', product_name), product_price);
run;

%put The price of Product A is: &price_ProductA;
```

This utilizes the `CATS` function to dynamically construct the macro variable name, creating a unique variable for each product's price.

Advanced Techniques and Considerations

Data Step Scope: Remember that `CALL SYMPUT` creates global macro variables, accessible throughout your SAS session.
Overwriting Variables: If a macro variable with the specified name already exists, `CALL SYMPUT` will overwrite its value.
Error Handling: While not explicitly part of the syntax, robust error handling should be

incorporated, particularly when dealing with potential issues like missing data or unexpected values.
Best Practices: Use descriptive macro variable names to enhance readability and maintainability. Always clearly document the purpose and usage of macro variables created using `CALL SYMPUT`.

Conclusion

`CALL SYMPUT` is a pivotal tool for building adaptable and efficient SAS programs. By dynamically creating and manipulating macro variables within the data step, it facilitates the development of complex data-driven applications. Mastering its functionality empowers users to automate tasks, simplify processes, and significantly enhance the flexibility of their SAS code. Understanding its intricacies, as detailed in this guide, is key to harnessing its full potential.

Frequently Asked Questions (FAQs)

1. Can I use `CALL SYMPUT` within a `PROC` step? No, `CALL SYMPUT` is specifically designed for use within the data step.
2. What happens if the value assigned to the macro variable is missing? The macro variable will be created but will contain a missing value. Proper error handling should account for this.
3. Can I use special characters in macro variable names created with `CALL SYMPUT`? While possible, it's generally best practice to avoid special characters to prevent potential conflicts and enhance readability.
4. How do I delete a macro variable created with `CALL SYMPUT`? Use the `%LET` statement with a null value: `%LET my_macro_variable=;`.
5. What is the difference between `CALL SYMPUT` and `%LET`? `%LET` creates macro variables within the macro language processor, while `CALL SYMPUT` creates macro variables within the data step, allowing for dynamic value assignment based on data processing.

Formatted Text:

how long is 30 cm

~~8 hours in minutes~~

65g to oz

56mm in inches

~~107 inch to cm~~

14 kg to lb

100 oz to gallons

720 mm in inches

140cm in feet

43kg in lbs

185 c to f

141 inches to feet

135kg to lb

51cm in inches

620 mm in inches

Search Results:

Solved: CALL SYMPUT vs CALL SYMPUTX - SAS Support ... 20 Jul 2017 · CALL SYMPUT() and CALL SYMPUTX() will convert numbers to text using BEST12. format. If you want to preserve the format attached to a number then either convert it to character yourself or use the VVALUE() (or VVALUEX()) function to get the formatted value.

call symput with a do loop - SAS Support Communities 23 Oct 2018 · Also make sure to use the newer CALL SYMPUTX() function instead of the old CALL SYMPUT() function. Unless you really NEED to store non macro quoted trailing blanks into your macro variables. 2 Likes

Call symput not working as expected - SAS Communities 11 Dec 2016 · Hi, I have the below call symput statements, when I run for the first time &st_dt. and &en_dt. wont resolve to actual values, but the second run resolves it. Could you please let me know why? First run has the warnings in the log saying that …

call symputx vs. symput - SAS Support Communities 4 May 2022 · CALL SYMPUTX uses a field width of up to 32 characters when it converts a numeric second argument to a character value. CALL SYMPUT uses a field width of up to 12 characters. CALL SYMPUTX left-justifies both arguments and trims trailing blanks. CALL SYMPUT does not left-justify the arguments, and trims trailing blanks from the first argument only.

%let VS call symput(s) - SAS Communities 27 Nov 2017 · with the call symput(s) : jour jour . It's the "**bleep**" when we use quotes in the %let , for example when the value must be contain spaces ! In the present case it's very dangerous , it's "better" to use call symput(s) . You own a solution to around this problem ?

[SAS □□□□□] □□□ □□ - CALL SYMPUT [SAS □□□□□] □□□ □□ – CALL SYMPUT □□□□□^^ □□ □□□□ SAS □ □□ □□□□ □□□□ □□□□ □□□□ □□□ □□□ □□□□ CALL SYMPUT macro routine□ □□ □□□□□ □□□□□. □□ CALL SYMPUT□ □□□ □□ □□□□□. CALL SYMPUT (“macro-variable”, value); CALL SYMPUT□ DATA step□□ value ...

Solved: Call symput - Macro date? - SAS Support Communities 16 Oct 2018 · Solved: Hello: I would like to have create a system today format with YYYY_MM_DD. Something wrong with my code below, could someone help me to fix

Use a macro variable created with the CALL SYMPUT routine inside ... 25 Apr 2023 · "The macro variable created by CALL SYMPUT can not be referenced inside the creating data step. But CALL EXECUTE, SYMGET and RESOLVE can be used to reference a macro variable with in the data step. " This is then followed up with "CALL EXECUTE statements get resolved first and then moved to the input stack while iterating the data step.

Solved: Understanding Call Symput! - SAS Support Communities 4 Apr 2013 · Re: Understanding Call Symput! Posted 04-04-2013 10:19 AM (4584 views) | In reply to robertrao Yes macro variables ID1 and ID2 with the values F and M respecfively.

Solved: When to use symput or symget? - SAS Communities 13 Mar 2012 · You use CALL SYMPUT (or better CALL SYMPUTX as it is more flexible) to create macro variables from data step values. You use SYMGET to get macro variable values. Usually you can just reference the macro variable instead. Here are some situations where you would want to ...

Call Symput

Mastering the Art of Call SYMPUT in SAS: A Comprehensive Guide

The SAS `CALL SYMPUT` procedure is a powerful tool for dynamic macro variable creation and manipulation. Unlike the `%LET` statement, which creates macro variables within the macro language processor, `CALL SYMPUT` creates macro variables within the data step, allowing you to leverage data step processing to define your macro variables. This flexibility proves invaluable for creating dynamic and adaptive SAS programs, particularly in scenarios where variable names or values need to be determined during data processing. This article will provide a comprehensive overview of `CALL SYMPUT`, exploring its functionality, syntax, usage scenarios, and potential pitfalls.

Understanding the Syntax and Functionality

The core syntax of `CALL SYMPUT` is straightforward:

```sas
CALL SYMPUT('macro-variable-name', value);
```

Here:

'macro-variable-name': This is a character string representing the name of the macro variable you wish to create or modify. It must be enclosed in single quotes.
value: This is the value assigned to the macro variable. It can be a numeric value, a character string (also enclosed in single quotes), or a SAS variable's value.

Crucially, `CALL SYMPUT` operates within the data step context. This means the macro variable's value is determined by the current row being processed. This opens up possibilities for creating macro variables based on aggregated data, conditional logic, or even iterating through datasets.

Practical Examples: Unveiling the Power of CALL SYMPUT

Let's explore several practical applications to illuminate the utility of `CALL SYMPUT`.

Example 1: Creating a macro variable based on a dataset's summary statistic:

Suppose we have a dataset `sales` with a variable `sales_amount`. We want to create a macro variable `total_sales` containing the sum of `sales_amount`.

```sas
data _null_;
set sales end=eof;
sales_sum + sales_amount;
if eof then do;
call symput('total_sales', sales_sum);
```

```
end;
run;

%put The total sales are: &total_sales;
```

This code calculates the sum and then uses `CALL SYMPUT` to assign it to the macro variable `total_sales`. The `%PUT` statement then displays the value of this newly created macro variable.

Example 2: Dynamically creating macro variables based on conditional logic:

Imagine we want to create a macro variable `highest_region` indicating the region with the maximum sales.

```sas
proc sql noprint;
select region
into :highest_region
from sales
group by region
having sum(sales_amount) = max(sum(sales_amount));
quit;

%put The highest selling region is: &highest_region;
```

While this example uses `PROC SQL`, the principle remains the same. The result of the SQL query is assigned to the macro variable `highest_region` using the implicit `CALL SYMPUT` functionality of `PROC SQL`'s `INTO` clause.

Example 3: Iterating through a dataset to create multiple macro variables:

Let's say we have a dataset listing product names and their prices, and we want to create a macro variable for each product's price.

```sas
data _null_;
set products;
call symput(cats('price_', product_name), product_price);
run;

%put The price of Product A is: &price_ProductA;
```

```

This utilizes the `CATS` function to dynamically construct the macro variable name, creating a unique variable for each product's price.

## Advanced Techniques and Considerations

Data Step Scope: Remember that `CALL SYMPUT` creates global macro variables, accessible throughout your SAS session.
Overwriting Variables: If a macro variable with the specified name already exists, `CALL SYMPUT` will overwrite its value.
Error Handling: While not explicitly part of the syntax, robust error handling should be incorporated, particularly when dealing with potential issues like missing data or unexpected values.
Best Practices: Use descriptive macro variable names to enhance readability and maintainability. Always clearly document the purpose and usage of macro variables created using `CALL SYMPUT`.

## Conclusion

`CALL SYMPUT` is a pivotal tool for building adaptable and efficient SAS programs. By dynamically creating and manipulating macro variables within the data step, it facilitates the development of complex data-driven applications. Mastering its functionality empowers users to automate tasks, simplify processes, and significantly enhance the flexibility of their SAS code. Understanding its intricacies, as detailed in this guide, is key to harnessing its full potential.

## Frequently Asked Questions (FAQs)

1. Can I use `CALL SYMPUT` within a `PROC` step? No, `CALL SYMPUT` is specifically designed for use within the data step.
2. What happens if the value assigned to the macro variable is missing? The macro variable will be
```

created but will contain a missing value. Proper error handling should account for this.
3. Can I use special characters in macro variable names created with `CALL SYMPUT`? While possible, it’s generally best practice to avoid special characters to prevent potential conflicts and enhance readability.
4. How do I delete a macro variable created with `CALL SYMPUT`? Use the `%LET` statement with a null value: `%LET my_macro_variable=;`.
5. What is the difference between `CALL SYMPUT` and `%LET`? `%LET` creates macro variables within the macro language processor, while `CALL SYMPUT` creates macro variables within the data step, allowing for dynamic value assignment based on data processing.

how long is 30 cm

3000 ft to miles

189lbs to kg

135 cm to inches

98 degrees fahrenheit to celsius

Solved: CALL SYMPUT vs CALL SYMPUTX - SAS Support ... 20 Jul 2017 · CALL SYMPUT() and CALL SYMPUTX() will convert numbers to text using BEST12. format. If you want to preserve the format attached to a number then either convert it to character yourself or use the VVALUE() (or VVALUEX()) function to get the formatted value.

call symput with a do loop - SAS Support Communities
23 Oct 2018 · Also make sure to use the newer CALL SYMPUTX() function instead of the old CALL SYMPUT() function. Unless you really NEED to store non macro quoted trailing blanks into your macro variables. 2 Likes

Call symput not working as expected - SAS Communities 11 Dec 2016 · Hi, I have the below call symput statements, when I run for the first time &st_dt. and &en_dt. wont resolve to actual values, but the second run resolves it. Could you please let me know why? First run has the warnings in the log saying that ...

call symputx vs. symput - SAS Support Communities 4 May 2022 · CALL SYMPUTX uses a field width of up to 32 characters when it converts a numeric second argument to a character value. CALL SYMPUT uses a field width of up to 12 characters. CALL SYMPUTX left-justifies both arguments and trims trailing blanks. CALL SYMPUT does not left-justify the arguments, and trims trailing blanks from the first argument only.

%let VS call symput(s) - SAS Communities 27 Nov 2017 · with the call symput(s) : jour jour . It's the "**bleep**" when we use quotes in the %let , for example when the value must be contain spaces ! In the present case it's very dangerous , it's "better" to use call symput(s) . You own a solution to around this problem ?

[SAS □□□□□] □□□ □□ - CALL SYMPUT [SAS □□□□□] □□□ □□ – CALL SYMPUT □□□□□^^ □□ □□□□ SAS □□□ □□□□ □□□□ □□□□ □□□□ □ □□ □□□ □□□□ CALL SYMPUT macro routine□ □□ □□□□□ □□□□□. □□ CALL SYMPUT□ □□□ □□ □□□□□. CALL SYMPUT (“macro-variable”, value); CALL SYMPUT □ DATA step□□ value ...

Solved: Call symput - Macro date? - SAS Support Communities 16 Oct 2018 · Solved: Hello: I would like to have create a system today format with YYYY_MM_DD. Something wrong with my code below, could someone help me to fix

Use a macro variable created with the CALL SYMPUT routine inside ... 25 Apr 2023 · "The macro variable created by CALL SYMPUT can not be referenced inside the creating data step. But CALL EXECUTE, SYMGET and RESOLVE can be used to reference a macro variable with in the data step. " This is then followed up with "CALL EXECUTE statements get resolved first and then moved to the input stack while iterating the data step.

Solved: Understanding Call Symput! - SAS Support Communities 4 Apr 2013 · Re: Understanding Call Symput! Posted 04-04-2013 10:19 AM (4584 views) | In reply to robertrao Yes macro variables ID1 and ID2 with the values F and M respecfively.

Solved: When to use symput or symget? - SAS Communities 13 Mar 2012 · You use CALL SYMPUT (or better CALL SYMPUTX as it is more flexible) to create macro variables from data step values. You use SYMGET to get macro variable values. Usually you can just reference the macro variable instead. Here are some situations where you would want to ...